UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/468,469	12/21/1999	REGINALD V. BLUE		3674

7590 03/14/2006

ROCCO L ADORNATO
UNISYS CORP
TOWNSHIP LINE & UNION MEETING ROADS
BLUE BELL, PA 19424

EXAMINER	
ALI, SYED J	
ART UNIT	PAPER NUMBER
2195	

DATE MAILED: 03/14/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Advisory Action Before the Filing of an Appeal Brief	**Application No.** _09/468,469_	**Applicant(s)** BLUE, REGINALD V.	
	Examiner Syed J. Ali	**Art Unit** 2195	

--The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

THE REPLY FILED 08 February 2006 FAILS TO PLACE THIS APPLICATION IN CONDITION FOR ALLOWANCE.

1. ☒ The reply was filed after a final rejection, but prior to or on the same day as filing a Notice of Appeal. To avoid abandonment of this application, applicant must timely file one of the following replies: (1) an amendment, affidavit, or other evidence, which places the application in condition for allowance; (2) a Notice of Appeal (with appeal fee) in compliance with 37 CFR 41.31; or (3) a Request for Continued Examination (RCE) in compliance with 37 CFR 1.114. The reply must be filed within one of the following time periods:

a) ☐ The period for reply expires ______months from the mailing date of the final rejection.

b) ☒ The period for reply expires on: (1) the mailing date of this Advisory Action, or (2) the date set forth in the final rejection, whichever is later. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of the final rejection.

Examiner Note: If box 1 is checked, check either box (a) or (b). ONLY CHECK BOX (b) WHEN THE FIRST REPLY WAS FILED WITHIN TWO MONTHS OF THE FINAL REJECTION. See MPEP 706.07(f).

Extensions of time may be obtained under 37 CFR 1.136(a). The date on which the petition under 37 CFR 1.136(a) and the appropriate extension fee have been filed is the date for purposes of determining the period of extension and the corresponding amount of the fee. The appropriate extension fee under 37 CFR 1.17(a) is calculated from: (1) the expiration date of the shortened statutory period for reply originally set in the final Office action; or (2) as set forth in (b) above, if checked. Any reply received by the Office later than three months after the mailing date of the final rejection, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

NOTICE OF APPEAL

2. ☐ The Notice of Appeal was filed on _____. A brief in compliance with 37 CFR 41.37 must be filed within two months of the date of filing the Notice of Appeal (37 CFR 41.37(a)), or any extension thereof (37 CFR 41.37(e)), to avoid dismissal of the appeal. Since a Notice of Appeal has been filed, any reply must be filed within the time period set forth in 37 CFR 41.37(a).

AMENDMENTS

3. ☐ The proposed amendment(s) filed after a final rejection, but prior to the date of filing a brief, will not be entered because

(a) ☐ They raise new issues that would require further consideration and/or search (see NOTE below);

(b) ☐ They raise the issue of new matter (see NOTE below);

(c) ☐ They are not deemed to place the application in better form for appeal by materially reducing or simplifying the issues for appeal; and/or

(d) ☐ They present additional claims without canceling a corresponding number of finally rejected claims.

NOTE: _____. (See 37 CFR 1.116 and 41.33(a)).

4. ☐ The amendments are not in compliance with 37 CFR 1.121. See attached Notice of Non-Compliant Amendment (PTOL-324).

5. ☐ Applicant's reply has overcome the following rejection(s): _____.

6. ☐ Newly proposed or amended claim(s) _____ would be allowable if submitted in a separate, timely filed amendment canceling the non-allowable claim(s).

7. ☒ For purposes of appeal, the proposed amendment(s): a) ☐ will not be entered, or b) ☒ will be entered and an explanation of how the new or amended claims would be rejected is provided below or appended.

The status of the claim(s) is (or will be) as follows:

Claim(s) allowed: _None_.

Claim(s) objected to: _None_.

Claim(s) rejected: _1 and 3-45_.

Claim(s) withdrawn from consideration: _None_.

AFFIDAVIT OR OTHER EVIDENCE

8. ☐ The affidavit or other evidence filed after a final action, but before or on the date of filing a Notice of Appeal will not be entered because applicant failed to provide a showing of good and sufficient reasons why the affidavit or other evidence is necessary and was not earlier presented. See 37 CFR 1.116(e).

9. ☐ The affidavit or other evidence filed after the date of filing a Notice of Appeal, but prior to the date of filing a brief, will not be entered because the affidavit or other evidence failed to overcome all rejections under appeal and/or appellant fails to provide a showing a good and sufficient reasons why it is necessary and was not earlier presented. See 37 CFR 41.33(d)(1).

10. ☐ The affidavit or other evidence is entered. An explanation of the status of the claims after entry is below or attached.

REQUEST FOR RECONSIDERATION/OTHER

11. ☐ The request for reconsideration has been considered but does NOT place the application in condition for allowance because: _____.

12. ☐ Note the attached Information Disclosure Statement(s). (PTO/SB/08 or PTO-1449) Paper No(s). _____

13. ☒ Other: See Continuation Sheet.

☒ attachment

MENG-AL T. AN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER

Continuation of 13. Other: Examiner notes that in the previous telephone communication with Applicant's representative, it was agreed that the Oliver reference did not appear to teach an object-oriented class object for implementing the read/write lock. However, upon closer consideration of Oliver, it became apparent that the reference does in fact teach such object-oriented methods for handling locks. That is, Oliver indicates that C++ class libraries are used for implementing the invention, and that the code was included in an attached appendix. Upon retrieving the appendix, it is clear that Oliver does in fact teach class objects and methods for performing all the functionality of the locks described in the specification. Therefore, the rejection must be maintained.

APPENDIX A

Page 1 of 5

```
// NAME
//    rwlock.cpp
//
// DESCRIPTION
//    Implementation of read/write locks under Win32
//
// COPYRIGHT
//    Copyright 1995 by Advanced Digital Systems Group, All rights reserved.
//

#include <windows.h>
#include <alttypes.h>


typedef HANDLE Handle;


inline void SyCloseHandle(Handle handle)
{
   if (handle != NULL && !::CloseHandle(handle))
      throw SySystemException();
}

inline Handle SyCreateMutex(LPSECURITY_ATTRIBUTES security, Bool8 initialOwner,
LPCTSTR name = NULL)
{
   Handle handle;
   if ((handle = ::CreateMutex(security, initialOwner, name)) == NULL)
      throw SySystemException();
   return handle;
}

inline Handle SyCreateSemaphore(LPSECURITY_ATTRIBUTES security, Int32 initialCount,
 Int32 maximumCount, LPCTSTR name = NULL)
{
   Handle handle;
   if ((handle = ::CreateSemaphore(security, initialCount, maximumCount, name)) == NULL)
      throw SySystemException();
   return handle;
}

inline void SyReleaseMutex(Handle mutex)
{
   if (!::ReleaseMutex(mutex))
      throw SySystemException();
}

inline void SyReleaseSemaphore
(
   Handle semaphore,
   Int32 releaseCount = 1,
   Int32* previousCount = NULL
)
{
   if (!::ReleaseSemaphore(semaphore, releaseCount, previousCount))
      throw SySystemException();
```

methods used by class

```
}

inline UInt32 SyWaitForMultipleObjects
(
   UInt32 count,
   const Handle* handles,
   Bool8 waitAll,
   UInt32 timeOutPeriod = INFINITE
)
{
   _ASSERT(count <= MAXIMUM_WAIT_OBJECTS);
   UInt32 index;

   if ((index = ::WaitForMultipleObjects(count, handles, waitAll, timeOutPeriod)) == WAIT_FAILED)
      throw SySystemException();
   if (waitAll)
   {
      if (index >= WAIT_OBJECT_0 && index < WAIT_OBJECT_0 + count)
         index = WAIT_OBJECT_0;
      else if (index >= WAIT_ABANDONED_0 && index < WAIT_ABANDONED_0 + count)
         index = WAIT_ABANDONED_0;
   }
   return index;
}

inline UInt32 SyWaitForSingleObject(Handle handle, UInt32 timeOutPeriod = INFINITE)
{
   UInt32 index;

   if ((index = ::WaitForSingleObject(handle, timeOutPeriod)) == WAIT_FAILED)
      throw SySystemException();
   return index;
}

//
// A read/write lock is not a primitive NT synchronization object.
// It can be owned by one writer or any number of readers, and is
// accessed through the following interface routines:
//
//    Handle SyCreateReadWriteLock(LPSECURITY_ATTRIBUTES security, Int32 initialCount)
//       // initialCount == -1 means it is initially owned by a writer
//       // initialCount == 0 means it is not initially owned
//       // initialCount > 0 means it is initially owned by that number of readers
//       // security defines the attributes for the primitive NT sync objects
//    void SyCloseReadWriteLock(Handle lockHandle)
//    Bool8 SyGetReadLock(Handle lockHandle, UInt32 timeout)
//    Bool8 SyGetWriteLock(Handle lockHandle, UInt32 timeout)
//       // the above two functions return TRUE unless the timeout expires without
//       // obtaining the lock
//    void SyReleaseReadLock(Handle lockHandle)
//    void SyReleaseWriteLock(Handle lockHandle)
//
//    // The following lower level functions allow the locks to be waited upon in
//    // conjunction with other NT synchronization objects.  Use one of the first two
//    // functions to get the handles for waiting on the lock, then use the handles
//    // in a WaitForMultipleObjects() call; waitAll must be TRUE to properly obtain
//    // the lock if numHandles is greater than one (currently a read lock uses one
//    // handle but a write lock uses two).  After obtaining the handles for the lock,
//    // the UpdateReadLock() or UpdateWriteLock() function must be called as soon as
//    // possible, and in all cases must be called before attempting to release the lock.
//    // After obtaining a lock in this fashion, it can be released with SyReleaseReadLock()
//    // or SyReleaseWriteLock().
//
//    const Handle* SyGetReadLockHandles(Handle lockHandle, UInt32& numHandles) const;
//    const Handle* SyGetWriteLockHandles(Handle lockHandle, UInt32& numHandles) const;
//    void UpdateReadLock(Handle lockHandle);
//    void UpdateWriteLock(Handle lockHandle);
//
// Note that the handle to the read/write lock cannot be used directly with any of
// the NT system wait functions (but see above), and cannot be passed to SyCloseHandle().
// Also, a read/write lock cannot be used across multiple processes.
//
// A write lock can be demoted to a read lock by calling SyGetReadLock() then calling
// SyReleaseWriteLock().  However, the converse cannot be done to promote a read lock
// to a write lock; a deadlock would occur.  A thread owning a read lock must release it
// before calling SyGetWriteLock().  There would be no real benefit to implementing
// read lock promotion over simply releasing the lock then attempting to get the write
// lock.
//

class SyReadWriteLock
{
   enum
   {
      kLockMutex,
      kWriteSemaphore,
      kNumHandles
   };
   Handle mHandle[kNumHandles];
   UInt32 mNumReaders;
   SyReadWriteLock(LPSECURITY_ATTRIBUTES security, Int32 initialCount)      constructor
   {
      mHandle[kLockMutex] = ::SyCreateMutex(security, initialCount < 0);
      mHandle[kWriteSemaphore] = ::SyCreateSemaphore(security, initialCount <= 0, 1);
      mNumReaders = initialCount > 0 ? initialCount : 0;
   }
   ~SyReadWriteLock(void)
   {                                                      destructor
      for (UInt16 i = 0; i < kNumHandles; i++)
         ::SyCloseHandle(mHandle[i]);
   }
   friend Handle SyCreateReadWriteLock(LPSECURITY_ATTRIBUTES security, Int32 initialCount);
   friend void SyCloseReadWriteLock(Handle lockHandle);
   friend Bool8 SyGetReadLock(Handle lockHandle, UInt32 timeout);
   friend Bool8 SyGetWriteLock(Handle lockHandle, UInt32 timeout);
   friend void SyReleaseReadLock(Handle lockHandle);
   friend void SyReleaseWriteLock(Handle lockHandle);
   friend const Handle* SyGetReadLockHandles(Handle lockHandle, UInt32& numHandles);
   friend const Handle* SyGetWriteLockHandles(Handle lockHandle, UInt32& numHandles);
   friend void SyUpdateReadLock(Handle lockHandle);
   friend void SyUpdateWriteLock(Handle lockHandle);
};
```

```
inline Handle SyCreateReadWriteLock(LPSECURITY_ATTRIBUTES security, Int32 initialCount)
{
   return new SyReadWriteLock(security, initialCount);
}

inline void SyCloseReadWriteLock(Handle handle)
{
   delete (SyReadWriteLock*)handle;
}

inline void SyUpdateReadLock(Handle lockHandle)
{
   SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
   if (lock->mNumReaders++ == 0)
      if
      (
         ::SyWaitForSingleObject(lock->mHandle[SyReadWriteLock::kWriteSemaphore], 0)
         !=
         WAIT_OBJECT_0
      )
         _ASSERT(FALSE);
   ::SyReleaseMutex(lock->mHandle[SyReadWriteLock::kLockMutex]);
}

inline void SyUpdateWriteLock(Handle lockHandle)
{
   SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
   ::SyReleaseSemaphore(lock->mHandle[SyReadWriteLock::kWriteSemaphore]);
}

inline Bool8 SyGetReadLock(Handle lockHandle, UInt32 timeout)
{
   SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
   const Handle* handles;
   UInt32 numHandles;
   handles = ::SyGetReadLockHandles(lockHandle, numHandles);
   switch (::SyWaitForMultipleObjects(numHandles, handles, TRUE, timeout))
   {
      case WAIT_OBJECT_0:
         ::SyUpdateReadLock(lockHandle);
         return TRUE;
      default:
         _ASSERT(FALSE);
         // drop through
      case WAIT_TIMEOUT:
         return FALSE;
   }
}

inline Bool8 SyGetWriteLock(Handle lockHandle, UInt32 timeout)
{
   SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
   const Handle* handles;
   UInt32 numHandles;
   handles = ::SyGetWriteLockHandles(lockHandle, numHandles);
```

```
    switch (::SyWaitForMultipleObjects(numHandles, handles, TRUE, timeout))
    {
       case WAIT_OBJECT_0:
          ::SyUpdateWriteLock(lockHandle);
          return TRUE;
       default:
          _ASSERT(FALSE);
          // drop through
       case WAIT_TIMEOUT:
          return FALSE;
    }
 }

 inline void SyReleaseReadLock(Handle lockHandle)
 {
    SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
    ::SyWaitForSingleObject(lock->mHandle[SyReadWriteLock::kLockMutex]);
    if (--lock->mNumReaders == 0)
       ::SyReleaseSemaphore(lock->mHandle[SyReadWriteLock::kWriteSemaphore]);
    ::SyReleaseMutex(lock->mHandle[SyReadWriteLock::kLockMutex]);
 }

 inline void SyReleaseWriteLock(Handle handle)
 {
    SyReadWriteLock* const &lock = (SyReadWriteLock*)handle;
    ::SyReleaseMutex(lock->mHandle[SyReadWriteLock::kLockMutex]);
 }

 inline const Handle* SyGetReadLockHandles(Handle lockHandle, UInt32& numHandles)
 {
    SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
    numHandles = 1;
    return &lock->mHandle[SyReadWriteLock::kLockMutex];
 }

 inline const Handle* SyGetWriteLockHandles(Handle lockHandle, UInt32& numHandles)
 {
    SyReadWriteLock* const &lock = (SyReadWriteLock*)lockHandle;
    numHandles = SyReadWriteLock::kNumHandles;
    return lock->mHandle;
 }
```